Stanford Artificial Intelligence Laboratory
Memo AIM-336

Computer Science Department
Report No. STAN-CS-80-796

March 1980

ESSENTIAL E

by

Arthur Samuel

Abstract

This is an introductory manual describing the display-oriented text editor E that is available on the Stanford A.I. Laboratory PDP-10 computer. The present manual is intended to be used as an aid for the beginner as well as for experienced computer users who either are unfamiliar with the E editor or use it infrequently. Reference is made to the two on-line manuals that help the beginner to get started and that provide a complete description of the editor for the experienced user.

E is commonly used for writing computer programs and for preparing reports and memoranda. It is not a document editor, although it does provide some facilities for getting a document into a pleasing format. The primary emphasis is that of speed, both in terms of the number of key strokes required of the user and in terms of the demands made on the computer system. At the same time, E is easy to learn and it offers a large range of facilities that are not available on many editors.

Essential E

Table of Contents

Preface 2

1. Introduction 3

2. Basic Features 3

3. A Typical Editing Session 9

4. Moving Around (while in the Normal Mode or with Attachments) 10

5. Editing Lines of Text (in the Line-Edit Mode) 12

6. Attaching and Copying Lines of Text 14

7. Search and Substitution Commands 15

8. Text Formatting Commands 18

9. Some Additional Commands 21

10. Macros (Saving and Repeating Command Strings) 25

Appendix A Missing and Equivalent DataMedia Keys 29

Appendix B File Switches 30

Preface

This manual is a condensation of the on-line E.ALS[UP.DOC] manual and it is not a substitute for the complete documentation therein presented. The aim is to provide the user with a hard copy manual that tells him what can be done by E and where he can find the more complete explanation in the on-line manual.

So many people have, by now, contributed to this editor that it is difficult to ascribe proper credit. Dan Swinehart, wrote the original TV editor and created the original version of ETEACH. Others who have worked extensively on E include Fred Wright, Brian Harvey, Arthur Samuel and Martin Frost. The present manual was written by Arthur Samuel.

Special Symbols

Special symbols are used in this manual (also in E.ALS[UP.DOC] and in ETEACH):

The ARROW LINE refers to one line of text that is indicated on the DataDisc terminals by the presence of an arrow to the left of this line, and on the DataMedia terminals by an underscore beneath the first character.

The CURSOR refers to the underscore under one character of the arrow line when in the LINE-EDIT mode. On the DataMedia terminals this cursor is distinguished from the cursor pointing to the arrow line by the fact that the entire line is brightened when in the LINE-EDIT mode.

The keys labeled "CONTROL" on the DataDisc terminal and "CTRL" on the DataMedia terminal are quite different and care must be taken not to confuse them.

On the DataDisc

α means hold the CONTROL key down while typing the next character.
β means hold the META key down while typing the next character.
$\alpha\beta$ means hold both the CONTROL and the META keys down while typing.
$\otimes$ means hold either the CONTROL key or both CONTROL and META keys with the next character.

On the DataMedia

α means hold the EDIT key down while typing the next character.
β means make an extra key stroke ahead of the character holding EDIT key and typing the NULL key.
$\alpha\beta$ means make an extra key stroke ahead of the character, holding EDIT key and typing the NULL key; then keep the EDIT key down while typing the next character.
$\otimes$ means do either as explained for α or as explained for $\alpha\beta$ (but not as for β).

Some of the characters on the DataDisc keyboard are missing from the DataMedia and require the use of the CTRL key. See the Appendix for full details.

Manual	Hold and Type		then Hold and Type	
αA	<EDIT>	A		(for a typical available key, represented by A)
βA	<EDIT>	<NUL>		A
$\alpha\beta$A	<EDIT>	<NUL>	<EDIT>	A
<CR>		<CR>		(for a typical special key, represented by <CR>)
α<CR>	<EDIT>	<CR>		
β<CR>	<EDIT>	<NUL>		<CR>
$\alpha\beta$<CR>	<EDIT>	<NUL>	<EDIT>	<CR>
ε	<CTRL>	F		(for a typical missing key ε)
$\alpha\varepsilon$	<EDIT><CTRL>	F		
$\beta\varepsilon$	<EDIT>	<NUL>	<CTRL>	F
$\alpha\beta\varepsilon$	<EDIT>	<NUL>	<EDIT><CTRL>	F

Essential E

1. Introduction

E is a display-oriented text editor available on the Stanford A.I. Laboratory PDP-10 computer. It is not a document editor, although it does provide some facilities for getting a document into a pleasing format. It is regularly used by many people for writing programs and for the preparation of the texts that are subsequently used as inputs to more elaborate document editors. The primary emphasis is that of speed, both in terms of the number of key strokes required of the user and in terms of the demands made on the computer system. At the same time, E is easy to learn and it offers a large range of facilities that are not available on many editors. E is an in-place editor and version copies are not made unless the user specifically arranges for this to be done.

This is an introductory manual, intended to be used as an aid for the beginner as well as for experienced computer users who either are unfamiliar with the E editor or use it infrequently. This manual is not intended as a replacement for the on-line E manuals but rather as a concise statement of the capabilities of the E editor that can be used to see what facilities E offers for handling specific tasks. In the interests of simplicity, most of the explanations will be in terms of the use of the DataDisc terminals with only occasional reference to DataMedia terminals.

Two on-line E documents are available.

ETEACH is an teaching program that guides the beginner, step by step, into the use of E. The only initial information needed is that of how to log on the system. Having logged on, the user simply types HELP ETEACH followed by a carriage return. The program then instructs the user, step by step. Even the experienced computer user may profit from a brief session with this program.

The second on-line document, E.ALS[UP,DOC], is an all-inclusive E manual. It can be referenced by typing ET?<CR> to the system or by typing <CONTROL>?, if currently editing some other file. Frequent references will be made to this on-line manual in the present publication. The directory on page 1 and the index of all commands on page 3 of this on-line document will be found most useful in locating additional information.

2. Basic Features

2.1 Paging

A typical E file is made up of a number of separate pages of differing lengths, where each page is restricted to some logically related sub-section of the total text. It is not necessary to adjust the page size to that desired for the finished document as there are a number of commands that allow one to break pages into smaller pages and to combine several small pages into one large page. It is also possible to bring several contiguous pages into core together when this is desirable. E functions best if the size of the individual pages is limited to perhaps 200 or 300 lines of text, although E is quite capable of handling very large pages indeed, but only at a substantial cost in terms of speed and of the load on the system.

Normally, only a single page is kept in core while it is being edited and changes in this page are not copied onto the disk unless requested or until one moves to another page or leaves the editor. This greatly decreases the demands on the system. It allows one to undo all of the recent corrections to the page being edited by a single command, if desired. It has the unfortunate effect that an entire page of recent corrections can be lost if the system should go down before the user has issued a save command or has done so indirectly by going to another page or another file or by exiting from

E. Because of paging, there is no practical limit to the size of the files that can be edited by E other than the limit imposed by the computer systems itself.

E provides special facilities for moving from page to page within any one file and for moving between files, with the information as to the recently visited files, including the page and line locations, kept in an internal stack so that one can return to these locations with a minimum of effort. One can move or copy text from any file so visited to other locations in this file or to any location in any other file.

2.2 File Formatting

E expects the file that is being edited to be formatted internally in a particular way. Please note that this file formatting has to do with the way that the file is stored in the computer and it has nothing to do with the text formatting per se. A file created by E will, of course, be automatically formatted. Should the user attempt to edit an unformatted file, E will prompt with the options that are available. While it is possible to use E with unformatted files, the user is strongly urged to allow files that occupy more than one page to be formatted. An unformatted copy can be made, if necessary, after editing has been completed.

Formatted files have a special first page that is created by E and automatically kept up to date as changes are made to the text. This directory page tells E where each page starts in the disk copy of the file, thus simplifying the work required for E to write or read a page of text. This first page also contains the text of the first line of each normal page. This text can be searched by special commands to locate the page on which some desired information may be found. There are special commands that will cause E to search through an entire page and list certain types of information on a newly generated first line to aid in this process. For texts where these special commands are not applicable, the user can still make this first page into an effective table of contents by manually putting the desired information on the first line of each page.

2.3 The Window

As mentioned earlier, E is a display-oriented editor. Since the screen size will not usually permit the display of an entire page of text, E exhibits the text that is being edited as if through a window, there being special single-character commands that move this window around on a page of the text. One line of text within the window, the ARROW LINE, is marked with an arrow to its left. More will be said about this later. Also, when editing a line of text, a CURSOR will underline one character in this ARROW LINE.

As an aid to the user, E provides certain bits of information on the screen, in addition to the text itself, that tell the user something about the file that is being editing and the state of the editing. This information is contained in a top and bottom 'border line', the top called the header line and the bottom called the trailer line. These lines may contain either a series of asterisks, indicating that there is no more text in the header or trailer direction, or a series of dots, indicating that there is some text in the indicated direction that is in core but that does not show in the window.

The header line also contains the following:

1 The number of the text line that appears just below the header line.
2 The number(s) of the in-core page(s).
3 The name of the file being edited.
4 Some information as to the editing mode (to be explained later).

The trailer line also contains:

1 The number of the current line (at the arrow) that will be entered if one starts to type.

2 The total number of lines on the current page.
3 The number of the current page.
4 The total number of pages in the file being edited.
5 Some information about the size of the page in core (do αβ ET? for more details).

In addition to the trailing line there is a region below it where typed-in information is echoed and where certain bits of information are typed out, either to inform the user of the status of his editing or to answer certain questions that he can ask.For example, the message "Rippling" may appear here to signify that E is readjusting the disk assignment of your file in response to an increase in the size of a page. Two commands (⊗XTERSE<CR> and ⊗XVERBOSE<CR>) enable the user to either restrict or expand the amount of information that E provides. There may also be two lines of information above the heading line (the WHO lines) that contain system information.

2.4 Special Keys

There are two rather special keys on the DataDisc keyboard. These keys, the CONTROL and META keys, are always used with other keys to control their meaning and they must always be depressed before, and held down during the striking of the associated key (just as one uses the SHIFT and TOP keys). In general, these special keys transform all of the normal typewriter keys into E commands. In this and other manuals these keys are indicated either by <CONTROL> and <META> in angle brackets, as here shown, or by a short-hand notation consisting of α for <CONTROL> and β for <META>, αβ for both <CONTROL> and <META>, and finally, ⊗ for those cases where it is optional to use either the <CONTROL> key alone or both the <CONTROL> and <META> keys (but not the <META> key alone). The EDIT key on the DataMedia keyboard takes the place of the CONTROL key but there is no equivalent key to the META key. Instead, one must precede the character in question with holding the EDIT key and typing the NUL key.

For example, to get αβ<CR> on a DataDisc, hold down the <CONTROL> and <META> keys then depress the <CR> key. To get this on a DataMedia, hold down the <EDIT> key, then depress the <NUL> key, and then, still holding the <EDIT> key, depress the <CR> key.

The exact meaning of the CONTROL and META keys if used separately or together varies with the conditions. *Remember:* do not type the character α or β, but rather use the special keys that are thus referenced.

Numerical arguments are frequently used with certain commands. Arguments will be indicated in this manual by the symbol # or by a typical numerical example. When the CONTROL and/or META keys are indicated, these keys must be held down for all digits of the number.

There are several other keys on the DataDisc keyboard that are used for special purposes by E. These will be referenced by their names in angle brackets, such as: <FORM>, <VT>, <BS>, <LINE>, <ALT>. The carriage return key (marked <RETURN> on the DataDisc and DataMedia keyboards and referenced in this manual as <CR>) also is used for several special purposes, as will be noted later. These keys are not held down while striking other keys although they may be used with the CONTROL and META keys. See Appendix A for the corresponding keys on the DataMedia.

2.5 To Start an Editing Session Using E

To start an editing session using E one types ET FOO<CR> where FOO is the name of an existing file. If one wants to create a new file named FOO, the system command is CET FOO<CR>. E allows files to be opened in either the READ-WRITE mode or in a READ-ONLY mode. One enters a file in the READ-ONLY mode by using ER instead of ET or by typing /R between the end of the name and the <CR>. The /R is called a switch, of which there are several, as will be explained later. The READ-ONLY mode should always be used if one wishes to read a file but not to make any changes to it.

When in this mode, edited changes may be made in the in-core copy but E will not permit this copy to be written out on the disk unless the mode is changed to READ-WRITE.

2.6 E Commands

Most of the commands used in E are single characters typed with the CONTROL and/or the META keys. These characters are usually mnemonics for the type of action that is being called for, but this is not always possible and some of them will simply have to be learned through frequent use. A few keys, such as the <CR> key, can be used as single-character E commands without the use of the CONTROL or META key. Other keys, if typed without the CONTROL or META key, will simply cause the indicated character to appear in the text, as explained in detail below.

Since there are many more commands than there are letters, some of the less frequently used (or more dangerous) commands require the typing of several letters. These commands all begin with ⊗X (for eXtend) followed by enough letters from the name (without special keys) to disambiguate the command. If not enough letters are typed, E will complain and ask you to retype the command. ⊗X commands are terminated by a carriage return, <CR> or in a few special cases by some other activating character. They often allow for extra parameters, to be typed between the name and the termination.

The <ALT> key performs a rather special function in E, that of aborting any partially-typed command. In general, when one discovers that one is typing the wrong thing, it is usually safer and easier to type <ALT> than to try some other means of correcting the error. In particular, <ALT> will: abort any partially typed ⊗X command; restore any line currently being edited in the LINE-EDIT or INSERT mode (to be discribed later) to its previous contents; terminate and abort any search or substitute command being typed.

Many E commands will accept a prefixed numerical argument, as will be explained in more detail later. It may be worth noting at this time that most of these commands may be made to report some information concerning the command, without actually executing it, by using a zero prefixed argument. This is often a very useful way of finding out what the command might do or what its current mode of operation may be. Information so reported is typed out at the bottom of the screen.

2.7 Editing Modes

The various options available to the user are grouped together into a few distinct modes. The default mode that E enters initially is called the NORMAL mode and this is the mode from which one enters most of the other modes. Many of the commands that may be used in the different modes will be described later. For the moment, it is important to recognize the existence of these different modes, since some commands have quite different actions in the different modes, and to learn a few of the commands that are so affected.

2.7.1 The NORMAL Mode

The NORMAL mode allows for movement between files, from page to page and within the page to arrive at any desired line. In normal editing, one will switch from this mode to other modes, as required, and will return to this mode frequently. Commands that affect, or relate to, an entire page or entire file are also executed in this mode. In fact, much of the power of the E editor resides in the large array of commands that are available in the NORMAL mode.

While in the NORMAL mode, an arrow appears on the screen, to the left of the text. The line on which the arrow appears, will be called the ARROW LINE. On some terminals, in particular on the

DataMedia, the ARROW LINE is identified by an underscore or CURSOR under the first character of the line.

Some special NORMAL mode commands should be learned at this time:

⊗4<CR>	Move the ARROW down 4 lines (but not beyond the trailer line).
α<CR>	No action when in this mode but used to return to it from other modes.
β<CR>	Introduce an empty line ahead of the ARROW LINE.
αβ<CR>	Enter the LINE-INSERT mode (see below) and introduce an empty line as the ARROW LINE.
D	Enter the LINE-EDIT mode and overwrite the first character in the ARROW LINE with a "D".
⊗3αD	Enter the LINE-EDIT mode and delete the first 3 characters in the ARROW LINE.
βD	Enter the LINE-EDIT mode and insert the character "D" ahead of the first character.
⊗6αβD	Delete 6 lines starting with the ARROW-LINE.

2.7.2 *The LINE-EDIT Mode*

This mode is used for making deletions, corrections and additions to text. It is frequently entered from the NORMAL mode simply by starting to type, using any one of the normal typewriter keys without any of the special control keys that are used to identify E commands. Many of the commands that are used to move around within a line of text may also be used to enter the LINE-EDIT mode, as will be explained later.

While in this mode, any typing that is done will appear in the ARROW LINE, and a CURSOR (underscore) will appear under the character position where the next typed character will be placed. On the DataMedia, the ARROW LINE will appear brighter than the other lines so as to differentiate the LINE-EDIT mode (where the CURSOR is used as just discribed) from the NORMAL mode (where the CURSOR is used to indicate the ARROW LINE).

The carriage return key (<CR>) is normally used to terminate the LINE-EDIT mode and to return to the NORMAL mode with the ARROW moved to the next line. It will perform this function even when the CURSOR is *not* at the end of the line being edited and the line will be written into the in-core copy in the same form as it appears on the screen. To insert a carriage return into the line at the CURSOR position (when in the LINE-EDIT mode) one must use the β<CR> command, as will be explained in detail later.

E does not automatically start a new line of text after some fixed number of characters (as some editors do), so the user must break his lines by typing carriage returns (or by any one of several special commands, as noted in section 5.3 below).

While E is capable of dealing with very long lines by use of some special commands, lines of text as typed must not exceed 133 characters in length. This limitation comes about because E makes use of the System Line Editor, which has such a limit. Offsetting this limitation is the feature that the line is actually copied whenever one enters the LINE-EDIT mode and the original [illegible]core line may be left unaltered by a single command (the <ALT> key) should one wish to abort the current changes at any time before typing the final <CR>.

2.7.3 *The INSERT Mode*

This mode is used when one wants to insert a substantial amount of new text in a line. It is entered from the LINE-EDIT mode or the NORMAL mode by typing αI. While in this mode, everything that one types (including carriage returns) will be inserted in the line. It also differs from the LINE-EDIT mode in that the newly typed text is inserted into the old text (that is, it does not overwrite), with the displaced text being moved along to make room. In this mode, the carriage return, <CR>, without control key, does not terminate the mode but instead causes a carriage return to be inserted into the text. This mode is usually terminated by typing α<CR>. The INSERT mode differs from the LINE-INSERT mode (to be discribed next) in that the <ALT> key may be used to restore the line to its original condition (but only back to the last typed <CR>, if one has been typed while in the INSERT mode).

2.7.4 *The LINE-INSERT Mode*

This mode is used when one wishes to insert one or more lines of text between existing lines. It is entered from the NORMAL and INSERT modes by typing αβI or αβ<CR>. While in this mode, everything that is typed (including carriage returns) will appear in the text. This mode is distinguished by the presence of the letter "I" in the header line and by the use of a double arrow in place of a single arrow in front of the ARROW line. It is terminated by typing α<CR> or <ALT>. The α<CR> command, if given at the beginning of a new line, will allow the empty line to remain in the text, while the <ALT> command will abort the empty line (but it will *not* abort a line containing text, as it does in the INSERT mode).

2.7.5 *The ATTACH mode*

While in this mode, a selected portion of the text is removed from the in-core text and is held in a special ATTACH buffer in such a way that it can be moved around within the opened file, or even moved to another file, by using the same commands that are used in the NORMAL mode. At all times, the text in the ATTACH buffer (or the first few and last few lines, if the selection is longer than a certain size) is shown on the screen at the position in the in-core text where it will be deposited should the appropriate release command be given. This mode is distinguished by the presence of the letter "A" in the header line and by vertical bars to the left of the attached text (brightening on the DataMedia). The commands for entering and leaving this mode will be described later.

2.8 Saving Corrections and Exiting from E

The command ⊗. causes the recently typed text to be written onto the disk. Please note that the period is the command. One should also note that the process of going to another page or of switching to another file also causes all corrections and additions to be written onto the disk.

To end the session and save recently made corrections type ⊗E. To end the session without saving the recently typed text, type ⊗XQUIT<CR>. One can, of course, exit by using the <CALL> key but this does not restore the screen as does the ⊗XQUIT command. After exiting, it is possible to reinstate the editor and redisplay the text by the system command CONTINUE. It is also possible to exit from E and go directly to certain other programs.

⊗XGO<CR>	Exit from E as with ⊗E and then repeat the last RPG command. This is an easy way to get back to a compiler after making corrections to a program.
⊗XRSYS <program><CR>	Exit from E as with ⊗E and run the system program as specified.
⊗XRUN <program><CR>	Exit from E as with ⊗E and run the user program as specified.
⊗XDRJ<CR>	Exit from E as with ⊗E and run the system program DIRED.

3. A Typical Editing Session

If you are a beginner at using E, a session with HELP ETEACH is recommended. If you have had some experience with either E or some other editor, the following brief summary of a typical editing session may be useful. References are given for locations where the detailed discription of each command (and related commands) may be found.

GET NEWFILE<CR>	This opens a new file named NEWFILE. (section 2.5 page 5)
⊗λOLDFILE<CR>	Entering this file in READ-ONLY mode. (section 4.2 page 11)
⊗P	This command might be repeated several times. (section 4.1 page 10)
<FORM>	This brings in another windowful. (section 4.1.2 page 10)
⊗≥	Move the ARROW down to the middle of the window. We will assume that this places the ARROW at a line in a paragraph that you want to copy. (section 4.1.3 page 10)
αβM	So mark this line. (section 4.3 page 11)
αH	Now back to NEWFILE. (section 4.2 page 11)

You are now typing into NEWFILE (in LINE-EDIT mode) and this line of text will appear exactly as here typed.

<CR>	Each carriage return that you type brings you back to the NORMAL MODE and moves the ARROW to the next line where you may continue typing. (section 5.3 page 13)
⊗?	Maybe a brief look at EALS[UP.DOC] might help at this point. (section 4.2 page 11)
αH	You are now back in NEWFILE so that you can continue typing as you wish. (A second αH would switch you to OLDFILE, not to EALS.)
⊗XMARK<CR>	It is time to start a new page. The first page will be automatically saved and you may continue typing into this second page. (section 8.1 page 18)
β<CR>	If given while in the NORMAL mode, this command will insert an empty line into the text ahead of the ARROW LINE. (section 2.7.1 page 7)
α<CR>	This enables you to finish typing or editing a line and return to the NORMAL mode with the ARROW pointing to the same line. (section 5.3 page 13)
αβM	Mark the line. (section 4.3 page 11)
⊗−⊗P	Going back a page to make some corrections. The second page will be saved.
⊗∞αI.	Move the ARROW to the end of the page. (section 4.1.3 page 10)
αβ4<BS>	Back up to the fourth line of text from the bottom. (section 4.1.3 page 10)
α<TAB>	Move the CURSOR to the end of this line. This command will enter the LINE-EDIT mode. (section 5.1 page 12)
<BS>	This backs up the cursor and deletes the last character. You can back up as much as desired and then start typing, ending always with a <CR>. (section 5.1 page 12)
αI.	Now back to the first line on this page. (section 4.1.3 page 10)
αβ2αβD	This deletes the first two lines on the page. (section 9.1 page 21)
⊗E	This ends the session. (section 2.8 page 8)

To Resume Editing if You Forgot Something.

CONTINUE<CR>	If this system command is given immediately after the ⊗E command everything will be restored just as if you had not given the ⊗E command, except, of course, the results of your earlier session will have been written onto the disk.
αH	This will still return you to OLDFILE.
αM	This returns you to the previously marked line.
α!⊗C	Attach a copy of the paragraph that contains the marked line. (section 9.5 page 21)
αH	Move this attachment to NEWFILE.
αM	And to the position where the mark in this file had been placed.
⊗E	Deposit the attachment at this point. (section 6.1 page 14)
⊗XJFILL<CR>	This page looks ragged since the copied text was formatted differently and you have been careless about line lengths, so adjust the line lengths for the entire page to be equal or less than the default value. (section 8.3 page 18).
⊗E	Now you are through. (section 2.8 page 8)

4. Moving Around (while in the NORMAL and ATTACH Modes)

Some of the more important commands for moving within a single file and from file to file will be listed here. For a complete list of these commands see E.ALS[UP,DOC].

4.1 Moving within a Given File

4.1.1 *Moving from page to page* (after writing out the current page to the disk):

⊗P	Go to the next page (note that ⊗+⊗1 is implied).
⊗6⊗P	Go to page number 6.
⊗−⊗P	Go back a page.
αO	Go to the previously viewed (Old) page as listed in the page stack.
αβO	Go to the second-previously viewed page as listed in the page stack. The meaning of this command can be changed by the user. For details see E.ALS.
⊗#αO	Go to the #th previous referenced page in the page stack.
⊗0αO	Type out the page stack (at the bottom of the screen) listing the more recently referenced pages, so that one may return to them directly.

4.1.2 *Window-moving commands (moving the ARROW, if necessary)*

<FORM>	Move forward one windowful but do not cross a page boundary.
<VT>	Move backward one windowful but do not cross a page boundary.
⊗<FORM>	Move forward one windowful even if this means writing out the page and crossing a page boundary.
⊗<VT>	Move backward one windowful even if this means writing out the page and crossing a page boundary.
⊗J	Jump the window so that the ARROW LINE is at the top of the screen..
⊗0⊗J	Jump the window to put the the ARROW LINE in the middle of the screen.
⊗−⊗J	Jump the window to put the line ARROW LINE at the bottom of the screen.
⊗T	Move the window up to show 4 more lines at the top.
⊗B	Move the window down to show 4 more lines at the bottom.

(see E.ALS for more)

4.1.3 *ARROW-moving commands (moving the window if necessary)*

<CR>	Move the ARROW down a line.
<BS>	Move the ARROW up a line.
⊗>	Move the ARROW down 4 lines.
⊗<	Move the ARROW up 4 lines.
⊗≥	Move the ARROW down 1/2 screen.
⊗≤	Move the ARROW up 1/2 screen.
⊗∧	Move the ARROW to the top line in the window.
⊗∨	Move the ARROW to the bottom line in the window.
αL	Move the ARROW to the first line of the current page.
αβL	Move the ARROW to the first line of the in-core text (there may be more than one page).
⊗#αL	Move the ARROW to the #th line of the page. Note that the line numbers do not appear on the text and they are not recorded, but E keeps track of them during the editing session so that they may be referenced.

αN Move the ARROW to the last referenced line as listed in the line stack. The line stack is cleared on changing pages.

αβN Move the ARROW to the second previously referenced line as listed in the line stack. The meaning of this command can be changed by the user. For details see E.ALS.

⊗#αN Move the ARROW to the #th previous line in the line stack.

⊗0⊗N Type the line stack (at the bottom of the screen).

4.2 Moving to Other Files

The task of moving back and forth between files is greatly simplified by the fact that E keeps a stack record of all recently visited files. Files once visited during any one editing session may be revisited by using one of the H (for Home) commands. File-switching commands that contain file-names may carry switches as defined in Appendix B. All file-switching commands write out the current page, if it has been modified, before changing files. Some useful commands are:

⊗εFOO<CR> Leave the present file and enter file FOO in the read-write mode (that is, so that it can be modified). The mode can be changed later by the command ⊗XREADONLY<CR>. The name of the file being left, together with the last referenced page and line number, will be placed in an internal file stack for use by the αH command, as noted below.

⊗εNEWF/C<CR> Leave the present file and enter a newly created file named NEWF.

⊗λFOO<CR> Leave the present file and enter file FOO in the read-only mode. The mode can be changed later by the command ⊗XREADWRITE<CR>.

⊗λLONGFILE/F<CR> Leave the present file and enter the file named LONGFILE, repaging and reformatting the core version only, without changing the disk copy. This is useful for examining an unformatted file with huge pages that has been produced by a program other than E.

⊗? Leave the present file and enter the E.ALS on-line-manual file, either at page 2 or, if it has been previously referenced during the session, at the last referenced position. The αH command will return to the present file without adding E.ALS to the file stack.

αH Return (Home) to the last referenced file at the page and line last referenced.

αβH Go to the second previously referenced file at the page and line last referenced. The meaning of this command can be changed by the user. For details see E.ALS.

α#αH Go to #th previous referenced file as listed in the file stack.

α0αH Type out (at the bottom of the screen) the file stack, listing the numbers that may be used to reference the files in the α#αH command. Note: the 0 shown in this command is the numeral zero, not the letter "O".

4.3 Aids in Moving Around

In addition to the file, page and line stacks that E maintains automatically, up to 23 lines in each referenced file may be marked by the user. These marks are saved and restored as one switches from file to file. Note that these marks are stored in the order of their appearance in the file so that they may be used to move through the file in an orderly fashion. The marks are *not* put into a stack as are the automatically-recorded αO and αN references.

αβM Mark the ARROW LINE, recording the page number and the line number.

αM Go to the next mark (cyclically).

⊗--αM Go to the previous mark (cyclically).

⊗#αM Go to the #th mark forward (cyclically).

⊗X/MARK<CR> Zero (clear) the MARKS in the current file.

5. Editing Lines of Text in the LINE-EDIT Mode

As mentioned earlier, deletions, corrections and additions to lines of text can be made using the LINE-EDIT mode. *While in this mode, any normal typing that is done will appear in the ARROW LINE and will overwrite characters that are in positions marked by the CURSOR.* It may be worth repeating at this point that the LINE EDITOR will only handle lines that contain less than 133 characters, so the user must terminate lines as they are being typed by using the <CR> key.

The LINE-EDIT mode is entered from the NORMAL mode either by typing any normal printing character (including the space bar and the tab key), or by typing any one of the special LINE-EDIT commands given below except for those marked *(will not enter)*. Commands that are so designated have a different meaning in the NORMAL mode from their meaning in the LINE-EDIT mode and so cannot be used to enter. It should also be noted that the LINE-EDIT mode cannot be entered while in the ATTACH mode. A limited number of NORMAL mode commands may, however, be given while in the LINE-EDIT mode.

5.1 Moving the CURSOR

α<SPACE>	Move the CURSOR to the right 1 character.
α4α<SPACE>	Move the CURSOR to the right 4 characters.
<BS>	Move the CURSOR to the left 1 character, deleting the character iff at the end of the line. *(will not enter)*
α<BS>	Move the CURSOR to the left 1 character but do not erase. *(will not enter)*
α5α<BS>	Move the CURSOR to the left 5 characters. *(will not enter)*
αβ<SPACE>	Move the CURSOR to the right continuously as long as the space bar is depressed (only repeats on the DataDisc).
α<TAB>	Move the CURSOR to the end of the line.
α<FORM>	Move the CURSOR to the start of the line. *(will not enter)*
αS *then* <char>	Move the CURSOR (Skip) to the next occurrence of <char>.
α3αS *then* <char>	Move the CURSOR (Skip) to the 3rd next occurrence of <char>
αB *then* <char>	Move the CURSOR Backward to the nearest occurrence of <char>. *(will not enter)*
α6αB *then* <char>	Move the CURSOR Backward to the 6th nearest occurrence of <char>. *(will not enter)*
αR	Repeat the last αS, αK, αB or αL command (saves retyping the <char>).

5.2 Inserting or Deleting Characters

β<char>	Make room and insert <char> ahead of the character that is in the CURSOR position.
αD	Delete the character at the CURSOR position and move the rest of the line to the left. If at the end of the line this will append the next line, but only if this does not make the line too long for the line-editor.
α4αD	Delete 4 characters starting at the CURSOR position (this will append the next line only if given at the end of the line).
αK<char>	Kill characters starting at the CURSOR position, up to but not including the first appearance of the character typed. Do nothing if no such character is found.
αK<CR>	Kill the rest of the line starting at the CURSOR position.
αL<char>	Kill backward starting to the left of the CURSOR, up to but not including the first encountered occurrence of the character typed. *(will not enter)*
αL<CR>	Kill backward everything before the CURSOR position to the start of the line. *(will not*

	enter)
αR	Repeat the last αS, αK, αB or αI: command (saves retyping the <char>).
αT	Transpose the order of the two characters just to the left of the CURSOR (to correct a common typing error). (*will not enter*)

5.3 Leaving the LINE-EDIT Mode

There are several commands for returning to the NORMAL mode. These may be given with the CURSOR at any position within the line and when so given a carriage return (actually a <CRLF>) will be placed at the end of the line; if the line is not already so terminated, and the line will be written into the in-core copy exactly as it appears on the screen.

<CR>	Accept the line, leave LINE-EDIT mode and move the ARROW to the next line.
α<CR>	Accept the line, leave LINE-EDIT mode but do not move the ARROW.
⊗U	Accept the line, leave the LINE-EDIT mode and move the ARROW Up one line.
<ALT>	Restore the line to its original condition and leave the LINE-EDIT mode.

One may also leave the LINE-EDIT mode and enter either the INSERT mode or the LINE-INSERT mode.

αI	Enter the INSERT mode (like holding <META> continuously). When in the INSERT mode, any LINE-EDIT command will return to the LINE-EDIT mode.
αβ<CR>	Break the line at the CURSOR and enter the LINE-INSERT mode with the CURSOR at the end of the first segment of the original line.
αβ↑	Enter the LINE-INSERT mode at the end of the previous line.
αβ↓	Enter the LINE-INSERT mode at the end of the following line.

5.4 Move to Another Line, still in the LINE-EDIT Mode.

These commands will accept numerical arguments, that must be given with αβ, to specify the number of lines to move in the direction indicated. The direction in which the semicolon and colon commands move can best be remembered by noting that they move in the same direction as indicated by the ↑ and ↓ symbols that are typed with these same keys.

α↑	Move to the end of the previous line.
α↓	Move the the end of the following line.
αβ:	Define the present column position as the COLUMN OF INTEREST and move to this column in the previous line (if possible, otherwise to the end of the line).
α;	Move to the COLUMN OF INTEREST (as earlier defined) in the previous line (if possible), without regard for the current column position.
αβ:	Define the present column position as the COLUMN OF INTEREST and move to this column in the following line (if possible, otherwise to the end of the line).
α:	Move to the COLUMN OF INTEREST (as earlier defined) in the following line (if possible), without regard for the current column position.

5.5 Duplicating a Line

⊗Q	Make a copy of the line just above the ARROW line and enter the LINE-EDIT mode in this newly created line. This is particularly useful if one needs a second line that is but slightly changed from an existing one.

6. Attaching and Copying Lines of Text

The Attach command removes the lines of text from the core image of the page and places them in a special attach buffer. The lines are still displayed on the screen in their original position (if there are more than a limited number of lines, then only the first and the last few lines are shown). The attached lines are marked with vertical bars at the left (except on DataMedias, where they are displayed in boldface). The apparent position of the attachment (and the place that it will be deposited) can be moved around with all of the usual page-moving, arrow-moving and file-switching commands (as described in Section 4 above). The Copy command first makes a copy of the desired text and then attaches this copy.

One note of caution: when moving a large portion of text from page to page, and particularly from file to file, it is safer to use the Copy command and then subsequently delete the original record than it is to use the Attach command, since the lines in the Attach buffer are not written on the disk and they could be lost if the system (or E) should malfunction while switching pages or files.

This is a good place to comment on the RIPPLING report that may appear on the bottom of the screen, particularly when moving text from page to page. This will occur whenever an increase in the size of a page requires that the disk assignments be altered.

6.1 Basic Commands

⊗A	Attach the ARROW LINE (removing it from the in-core text).
⊗C	Copy and attach the copy of the ARROW LINE. If this command is given when there is already something in the attach buffer, then the entire attachment will be deposited and recopied. This provides a convenient way to make multiple copies of any desired portion of text.
⊗4⊗A	Attach 4 lines starting with the ARROW LINE.
⊗2⊗C	Copy and attach the copy of 2 lines starting with the ARROW LINE.
α!⊗A	Attach the current paragraph. See Section 8.6 for more details.
αβ!⊗A	Attach the rest of the current paragraph starting with the ARROW LINE.
⊗E	Deposit the attached lines into the core image at the location shown on the screen and return to the NORMAL mode. Caution: striking E a second time will write out the page and leave E, so give the system time to react.
αβR	Replace the attached lines in their original position. This is extremely useful if you find that you are moving the wrong lines and you are not sure where you got them.
αβK	Kill the attached lines. Note: αK does not work. If one makes a practice of using the control key alone for those cases where it will work, one may avoid the danger of accidentally killing an attachment.
αβ4αβK	Kill the first 4 lines in the Attach buffer. (See ⊗−⊗C below.)

6.2 Commands with Relative Arguments (to add or remove lines from the end of the attach buffer).

All of these commands will also accept numerical arguments. Note that these commands allow one to assemble text abstracted from different locations and indeed from different files for eventual depositing at a single location.

⊗+⊗A	Attach one more line to the end of the attach buffer.
⊗+⊗3⊗A	Attach three more lines to the attach buffer.
⊗−⊗A	Detach one line from the end of the attach buffer.
⊗+⊗C	Copy one more line and add it to the end of the attach buffer.
⊗−⊗C	Delete (kill) one line from the end of the attach buffer.
⊗−⊗3⊗C	Delete (kill) three lines from the end of the attach buffer.

7. Search (FIND) and Substitution Commands

E provides a fairly elaborate set of search and substitution commands that can be applied to a single page or to an entire file.

Single page searches use the ⊗F command, while multi-page searches use the extended command ⊗XFIND. In both cases the command is executed starting with the ARROW LINE and extends either to the end of the current page or to the end of the file. When the current ARROW LINE has been reached as the result of an earlier search, the new search will begin just beyond the first character of this earlier found occurrence of the searched-for string. Multi-page searches may be terminated at the end of the page that is currently being searched by giving the command <ESC>I (the page number of the last-searched page will be reported).

Two commands, ⊗#⊗XSSLINE<CR> and ⊗#⊗XSSPAGE<CR> are available to set the search limit, i.e., to stop the search before the specified line or page for all subsequently issued commands (except for directory searches).

If the search is to be for a delimited string (*i.e.*, a portion of text that is separated from the rest of the text by spaces, tabs, <CR>s or other special characters), both the <CONTROL> and the <META> keys are used, while only the <CONTROL> key is used when initiating a non-delimited search. This is an important distinction.

The default form of ⊗F and ⊗XFIND is to consider upper and lower case letters to be equivalent but the command ⊗XEXACT<CR> can be used to force a case distinction for all subsequent searches (with ⊗−⊗XEXACT<CR> to undo the distinction).

There are several symbols that have special meanings when used in the searched-for string, as shown in the following table. If these characters are desired as ordinary characters in the searched-for string they must be preceded by the identity symbol ≡.

Symbol	Meaning
∀	Any character may appear in this position.
¬	Any character except for the one that follows this symbol may appear.
|	Any delimiter (not a letter, number or $ % or _) may appear.
≡	Treat (quote) the (normally special) symbol that follows as an ordinary character.
∞	Not now used but must be quoted.
⊂	Not now used but must be quoted.

Finally, the action to be taken on the successful completion of the search may be specified by the delimiter that is used to terminate the searched-for string. Commonly used delimiters are: <CR>, which calls for the ARROW to be move to the first line in which the searched-for string is to be found, and α<CR>, which causes the LINE-EDIT mode to be entered with the CURSOR positioned under the first letter of the searched-for string. Other delimiters are described in Section 7.1 below. Should the search be unsuccessful, no action is taken, the ARROW LINE remains undisturbed and the failure is reported.

A few examples follow.

αFSmith<CR> Find the first occurrence of the string *SMITH* or *smith* or even *smith* in *blacksmith* or in *Smithsonian* on the current page, starting with the present ARROW LINE, and move the ARROW to this new line. There is to be no space after the F since this is a single-letter command.

αXF smith<CR>	Same as above but searching from the ARROW LINE to the end of the file. Note the space after the F. Also note that currently there are no other word commands starting with F, so only the F of FIND need be typed, but there must be a single space after the F in this case to signal the end of the word.
αXF smithα<CR>	As above but enter the line editor with the CURSOR under the first letter of the searched-for string.
αβFsmith<CR>	Find the delimited string *smith*. This would *not* find smith in blacksmith or in Smithfield.
⊗9αβFsmith<CR>	Find the 9th occurrence of the string. If there are fewer than 9 occurrences report the number and do not move the ARROW. Note: asking for an impossibly large number (such as ⊗∞) is a convenient way to have the occurrences counted.

7.1 Search-String Terminators

It will be possible to list but a few of the more important search-string-terminating commands. Remember, these terminators are to be used as a part of a search command. They have quite different meanings when typed separately.

<CR>	Move the ARROW to the line found or report failure.
α<CR>	Move the ARROW to the line found, enter the line editor and place the cursor under the first character of the searched-for string.
⊗J	Jump the line with the searched-for string to the top of the window.
⊗A	Attach all lines starting with the ARROW LINE up to and including the line in which the searched-for string is found. If this termination is used with the extended search command and if the string found is not on the in-core page(s), the attachment will start at the beginning of the page containing the string.
αβD	Delete all lines up to but not including the line containing the searched-for string. This is a very dangerous command so use it with extreme caution.
αP	Make the search over the directory page (but without moving the window to it) and then display the referenced page with the ARROW pointing to the first line on this page.
αβP	Make the search over the directory page (but without moving the window to it) and then display the referenced page with the ARROW pointing to the second occurrence on this page of the searched-for string (the first being in the top line).
⊗+⊗P	Just like ⊗P except that the search starts with the next page after the current one.
⊗:	Make the search over the directory page and then display the referenced page with the ARROW pointing to the line containing the referenced string as a label (see E.ALS for details).
⊗\	Terminate the searched-for string and prepare to accept a substitution string. See section 7.3 below.
<FIND>	Like ⊗\ for use on terminals without CONTROL and META keys.
<ALT>	Abort the present search command that is in process of being typed.

7.2 Repeating Search Commands

The last-used search strings are remembered by E (that is, they are sticky). For a repeat, one can type the rest of the command including the desired termination (which may differ from the termination used the first time), but omitting the string and the previously defined string will be used. The search string for a within-a-page search and the search string for an extended search are remembered independently.

There are even simpler commands to ask for a repeat of the last search.

α∗ Find the next occurrence of the string specified by the last search command (whether a single-page or a multi-page search). Leave the line in the LINE EDITOR.

αβ∗ Find the next occurrence as above but do not enter the LINE EDITOR.

7.3 Substitution Commands

Substitution commands are initiated by first giving a normal search command that is terminated with ⊗\ (or <LINE>) and following this with the string to be substituted, exactly as one wants it to appear. The substitution string, in turn, is terminated by a <CR> or a α<CR> and this can have a prefixed numerical argument which specifies how many substitutions are to be made (see examples below). Note this distinction: a prefixed argument to the search part specifies where the substitution is to start while the argument before the final termination specifies the number of substitutions that are to be made. Substitution strings, so specified, are sticky and need not be retyped for repeat commands. Null substitutions are allowed, so be careful. Type <ALT> to abort a partially typed search or substitution command.

If several identical substitutions are to be made, it is usually wise to make a single substitution initially, using the α<CR> termination, and then to make the rest of the substitutions using the repeat command ⊗#αβ\ after checking that the first desired substitution has indeed been made correctly. This procedure is especially recommended when substituting a null string! The infinity sign (∞) can be used as the argument if substitutions are to be made for all occurrences.

Sample commands are:

αβFstringaαβ\Stringb<CR> Find the first occurrence of stringa (or Stringa) and replace it by Stringb, where the indicated upper and lower cases in the substitution string are to be as typed. Do not leave the line in the LINE EDITOR. Note: this is faster than with the α<CR> termination but the substitution is not cancellable with <ALT>.

αβFstringaαβ\Stringbα<CR>. As above but leave the line in the LINE EDITOR with the CURSOR under the last character of the substituted string. This is slower than the α<CR> termination but it has the advantage that the substitution may be cancelled by the <ALT> command. Use this command if you are not certain that you have the correct substitution. It is also good practice always to make a single substitution first and then to use the repeat command, shown below, for other desired substitutions after verifying that the desired substitution has, in fact, been made.

⊗9αβFSTRINGAαβ\STRINGBαβ∞<CR> Find all occurrences of the string STRINGA and start the substitutions of STRINGB for STRINGA at the 9th occurrence.

α\ Repeat the previously-made substitution at the next occurrence of the searched-for string, leaving the line in the LINE EDITOR with the cursor under the last character of the substituted string. Note the α\ requires no other termination in this case.

αβ\ Repeat the previously made substitution at the next occurrence of the searched-for string but do not leave the line in the LINE EDITOR. This is faster than the α\ command.

⊗6αβ\ Repeat the previously-made substitution for the next 6 occurrences. One should always use αβ\ as here shown and not the α\ command which causes a lot of useless entering and leaving of the LINE EDITOR. On the completion of the last substitution, the α<CR> command will cause the LINE EDITOR to be entered and the CURSOR to be moved to the last character of the last substitution, if this is desired.

⊗∞αβ\ Repeat the substitution until all occurrences of the searched-for string have been replaced (within the range specified in the original substitution).

⊗0⊗\ Report the last-used searched-for and substitution strings.

8. Text Formatting Commands

Text formatting commands in E are given while in the NORMAL or ATTACH mode. They usually apply to the specified number of lines or, by default, either to an entire page (or ATTACH buffer) or to a single line as will be noted later. Commands given in the ATTACH mode always apply to the contents of the ATTACH buffer.

8.1 Page Appending, Dividing and Combining Commands

⊗XAPPEND<CR>	Append the next page to the page (or pages) that are in core so that the two or more pages may be viewed and edited togather. The in-core page mark will be represented on the screen by a line of vertical bars.
β<FORM>	Split the current page into two pages with the ARROW LINE as the first line of the newly created page and retain both pages in core without writing out anythin The in-core page mark will appear as a line of vertical bars.
⊗XMARK<CR>	Split the current page as above but write out the in-core pages. Retain in core only the new page, now starting at the ARROW LINE. If this command is given in the ATTACH mode, the attached lines are first put down and then the page is divided. If the attachment is not at the top of a page the division is at the beginning of the former attachment. If the attachment is a the top of the page, the division is at the end of the former attachment. In either case, the ARROW is at the top line of the former attachment and only this page is retained in core.
⊗XDELETE<CR>	Delete the page mark at the end of the last in-core page and add the text of the next page to the last in-core page, writing out the resulting text.
αβD	When given with the ARROW pointing to an in-core page mark, delete this page mark without writing out the resulting text.

8.2 Line Breaking and Joining Commands (particularly useful for lines without word separators and for lines that are too long for the LINE-EDITOR)

⊗XBREAK 60<CR>	Break the ARROW LINE (once) after the 60th column (the initial default column is 80). A preceding numerical argument of one or larger will cause that many lines to be broken as many times as necessary (see the next command).
⊗4⊗XBREAK<CR>	Break 4 lines starting with the ARROW LINE as many times as necessary so that none of the resulting lines are longer than the default length.
⊗4⊗XJOIN<CR>	Join the 4 lines starting with the ARROW LINE into one line by removing the CRLFs. The default argument for this command is 2 lines.

8.3 Line Filling and Justifying Commands (for use with normal text)

Only the simpler forms of these commands will be described in this manual. E actually provides a wide range of commands for justifying text and tables. See the EALS manual for details. If the text is already broken into paragraphs that are separated by blank lines then the default options may be used. It is usually wise to save any recent corrections by doing a ⊗. before using these commands so that one can undo their effect with the ⊗XCANCEL<CR> command if necessary. The ⊗XUNDELETE<CR> command may also be used to undo the effects of these commands.

All of these commands can be preceded by a numerical argument to define their range of action. If given when there is text in the ATTACH buffer, they will apply to the attachment only. The ⊗! prefix is particularly useful in defining the range, as will be noted below. If the range is not otherwise specified, the command applies to the entire page or to the entire ATTACH buffer.

These commands will accept a string of numbers (separated from the command name by a space and from each other by commas) that are used to specify the justification margins. These values will then replace the default values and will apply to all subsequent filling and justifying commands. Normally 4 such values are specified, these being the indentation of the first line of each paragraph (Crown margin), the indentation of all subsequent lines (Left margin), the maximum length of the line including the indentation (Right margin) and the number of blank lines desired between paragraphs (Blanklines margin, where positive numbers refer to the number of blank lines and −1 specifies that the actual number of blank lines used in the original text are to be preserved). Only the values that are to be changed need be specified but commas must be typed to indicate any missing values that precede the values that are typed. The initial default values for these margins are 0,0,74,−1 (C,L,R,B).

Paragraphs are normally detected in the input text by (1) an empty line or (2) a line with exactly the specified Crown indentation. Under some circumstances it may be necessary to set the Detect-Paragraph margin to a different value from the desired new Crown indent. This can be done by typing the desired value followed by a slash at the start of the margin specifying string. See E.ALS for full details.

It is possible to use the ⊗XJGET<CR> command, as described below, causing E to get the margin values from in-core paragraphs.

⊗XJFILL<CR> Adjust the length the text lines to be equal to or less than the currently specified value by moving words from line to line within the individual paragraphs, fixing the left margins to the currently specified value.

⊗XJUST<CR> Adjust the line lengths as with ⊗XJFILL<CR>, but then pad the lines by distributing additional spaces between the words so as to align the right margins.

⊗XJFILL 4,0,66,0<CR> Reset the margin values before executing the command as defined above. The Crown margin will be 4, the Left margin will be 0, the Right margin will be 66 and there will be no blank lines between paragraphs.

⊗XJFILL 4/0,4,66,0<CR> Reset the margin values before executing the command. This is the form that the command would have to take if one wished to reverse the Crown and Left margin values for text that had previously been ⊗XJFILLed with the command listed above.

⊗4⊗XJFILL<CR> Adjust the length of the next 4 text lines (the count applies to the lines as they are initially) starting with the ARROW LINE.

⊗4αβ!⊗XJFILL<CR> Starting with the ARROW LINE, apply the command to 4 paragraphs. Note that the rest of the paragraph containing the ARROW LINE counts as the first paragraph. If the ! is used without an argument, it is taken to mean one paragraph. See section 8.5 below for more details on the paragraph designating symbol !.

⊗4α!⊗XJFILL<CR> As above, but starting at the beginning of the paragraph containing the ARROW.

⊗0⊗XJFILL<CR> Report the current margins without any overt action other than to record the typed changes to these values that may accompany the command (and that are recorded before the report is made). The astute user can reacquaint himself with the details of the way in which these values are assigned by using this and the following command.

⊗#⊗XJGET<CR> Scan the specified number of text lines to determine the indent values for the first and subsequent lines in each paragraph and the column position of the rightmost character. Store these for use with subsequent commands. This command is handy if the user has already typed one or more paragraphs in the desired format (perhaps in an earlier editing session) and wishes to set the justification parameters to these former values.

⊗XSJFILL<CR> Separate the sentences in the specified text so that they all start on separate lines. This is useful whenever it becomes desirable to rearrange the sentences within a paragraph or to move sentences between paragraphs.

8.4 Aligning, Indenting and Centering Commands

These commands preserve the identity of the text lines and insert or delete leading spaces and tabs as required. In contrast with the justifying commands, these commands default to operate on a single line (the ARROW LINE). A prefixed decimal argument or the use of the ATTACH buffer may be used to extend their range of application. If the text contains interior tabs, these are converted to spaces before the move, so that the absolute size of the spacings are preserved. An option exists, as demonstrated below, for changing the default condition so as to preserve interior tabs. When possible, initial tabs are used to save disk storage space. A zero preceding argument will cause the existing default values to be displayed..

⊗XALIGN<CR>	Align the ARROW LINE (or the entire ATTACH buffer), moving the line or lines so that they start with the default align indentation (originally set to 0).
⊗8⊗XALIGN 3<CR>	Align 8 lines starting with the ARROW LINE and indent them by 3 spaces.
⊗8⊗XALIGN T 3<CR>	As above, but preserve interior tabs. Note that the T argument is sticky. The default condition can be restored by using the letter S instead of T.
⊗→	Move the ARROW LINE to the right by the (absolute) default-indent value (initially at 4).
⊗5⊗→	Move 5 lines starting with the ARROW LINE to the right by the default amount.
⊗←	Move the ARROW LINE to the left by the (absolute) default value. The movement is limited by the first non-blank character in the line.
⊗XINDENT 8<CR>	Set the default indent value to 8 and move the ARROW LINE (or the entire attach buffer) to the right by 8 spaces. Note that the ⊗→ and the ⊗← commands still move the text in the indicated direction even when this command has been used to set the default to a negative value..
⊗XCENTER<CR>	Center the ARROW LINE between the left and right margins as defined by the current justification margins.

8.5 Formatting Tabular Material

Formatting tabular material is dificult if (1) the initial material is improperly aligned in columns or (2) if there are missing entries. E provides facilities for handling two distinct types of tabular material. In the first type, there can be no missing entries but the material need not be properly aligned into columns as long as the different entries are separated from each other by at least 2 spaces or by tabs (and, of course, the entries themselves must not contain double spaces or tabs), and the input material may contain some associated text (perhaps a comment field) that is to be justified or jfilled. In the second type, the original material must be aligned in columns but there can be missing entries.

Table-formatting commands require the specification of additional values beyond the C,L,R,B values specified with the normal-text-formatting commands. When these are typed they must start with a semicolon and they are separated by commas. Values not otherwise marked are for field lengths, with the first field assumed to start at the Left margin. For example, ;5,8,9<CR> means that the first field occupies 5 character positions, the second occupies 8 positions and the third occupies 9 positions. For more details see E.ALS. A command ⊗XTGET<CR>, analogous to ⊗XJGET<CR> is available for establishing the initial conditions from a line in the table that does not have missing entries.

For details concerning the ⊗XTABLE<CR> command for material already aligned in columns and the ⊗XJUST<CR> and ⊗XFILL<CR> commands for misaligned tabular material, see E.ALS.

9. Some Additional Commands

8.1 Line Deleting and Undeleting Commands

αβD	Delete the ARROW LINE.
⊗4αβD	Delete 4 lines starting with the ARROW LINE.
⊗XCANCEL<CR>	Restore the in-core copy of the current page from the disk, thus cancelling all changes made since the last disk-save command.
⊗XUNDELETE<CR>	Restore to the beginning of the ATTACH buffer the last group of lines altered or deleted by a single command. Note that any lines changed by the last-executed line-changing command may be restored by this command.
⊗#⊗XUNDELETE<CR>	Restore the last # number of lines whether or not these were all altered or deleted by a single command.
⊗0⊗XUNDELETE<CR>	Report the number of lines that are available to undelete. See E.ALS[UP.DOC] for more details.

9.2 Switching to a File Named on the Page Being Edited

⊗XPOINTER<CR>	Switch to the first file named in the text being edited at or below the ARROW line. See E.ALS[UP.DOC] for details as to how the file name must be formatted to be recognized by this command.

9.3 Conserving File Space

⊗XSIN<CR>	Put spaces in place of tabs on the current page or the attach buffer. This will, of course, increase the space occupied by the text but it may be necessary to preserve spacings if extra characters are to be added ahead of the tabs. This command should be followed by the ⊗XTIN<CR> command and then the ⊗XBURP<CR> command as soon as possible.
⊗XTIN<CR>	Replace spaces by tabs where this will reduce the space occupied by the text. This command also deletes trailing spaces and tabs from all lines. Note that this extra space is all collected as all-null records at the end of the page and that the ⊗XBURP<CR> command is still needed to save this space on the disk.
⊗XBURP<CR>	Write out the current page and in the process reduce the disk space for the page by deleting any all-null records from the page.
⊗ε/∞F<CR>	Burp the entire current file, not just the current page.

9.4 Generating a New Directory Line

⊗XNDFAIL<CR>	Generate a New Directory line for the current page for a FAIL- or MACRO-type source file.
⊗XNDSAIL<CR>	Generate a New Directory line for the current page for a SAIL-type source file. This command can be modified for use with PASCAL or any similiar language by replacing or adding to the list of category words that are used to signal the inclusion of the word that follows the category word. Some minor hand editing may be required. See E.ALS[UP.DOC] for details.

9.5 Message and Paragraph Designating Commands

The partial character ∂ on the DataDisc keyboard and the explanation point !, when used with ⊗, refer to a message and a paragraph, respectively. A message is defined as a contiguous set of lines (some of them may be blank) which starts with a ∂ and which is followed either by a line beginning with another ∂ or the end of the page. A paragraph is similarly defined as a contiguous set of non-empty lines plus a single preceding empty line, if one exists, and all but one of any following empty lines (actually all of the following empty lines if there are no more non-blank lines on the page).

These two prefixes, by themselves, usually do not initiate any action but they do define the range of action of the command that follows. The one exception in which the ∂ or ! initiates an action without waiting for a following command is when it is preceded by a 0 argument (that is ⊗0⊗∂ and ⊗0⊗!). For these cases the command immediately moves the arrow to the beginning of the current message or paragraph.

Typing the ∂ or ! with <CONTROL> only, defines the message or paragraph as starting at the beginning of the current message or paragraph even if the ARROW is not at the beginning. Typing the ∂ or ! with both <CONTROL> and <META> defines the range of action as starting at the ARROW LINE.

The ∂ command has an added property not possessed by the ! command. This property is that it will cause the deletion of an entire page (including a page mark) if the command is to delete or attach an entire message and iff the message is the only message on the page and there are other pages in the file. The next page (if there is one) or (if not) the preceding page will be brought into core.

Some typical commands are:

⊗0α∂	Move the ARROW to the beginning of the current message.
α∂⊗A	Attach the current message (at the end of the attach buffer if there is already something attached).
⊗--α∂⊗A	Attach the previous message (at the beginning of the attach buffer ahead of material that may already be attached).
⊗3α∂αβD	Delete 3 messages beginning with the current one (stopping if the last message on the page has been deleted).
⊗−⊗∞α∂αβD	Delete all messages in the in-core pages that preceed the current one.
⊗!αβ<CR>	Enter the LINE INSERT mode between the current paragraph and the next one.
α!⊗XJUST<CR>	Justify the current paragraph.
αβ!⊗XJUST<CR>	Justify the rest of the current paragraph starting with the ARROW LINE.

9.6 Parenthesis and Bracket Finding and Matching Commands

Several commands are available for finding and matching parentheses, both of the conventional sorts and using any desired set of (single) characters, as defined by the user. The default set is, of course (and). Some users find it convenient to use the set ⊂ and ⊃ to mean *begin* and *end* while writing programs so that the parenthesis matching commands may be used to keep track of nesting levels, and then substitute *begin* and *end* for these only when required to do so by the limitations of the compiler (PLUG: the SAIL compiler does not suffer from such a limitation).

Some useful commands are:

⊗(	Find the next left symbol of either the default variety or as defined by a previous ⊗XPAREN command (see below), limiting the search to the in-core page(s). If the CURSOR is already at a left symbol this command is a NO OP. The convention is that when the CURSOR is under a left symbol, it refers to the condition just to the left of this symbol. Similarly, when the CURSOR is under a right symbol, it refers to the condition just to the right of this symbol.

⊗4⊗(Move the CURSOR to that left symbol that would cause the nesting level to reach 4, considering the present nesting level to be 0. Do not move the CURSOR if this level is never reached.

⊗4⊗XLPAREN<CR> Similar to the above except that the search is not limited to the in-core page(s).

⊗) If the CURSOR is under a left symbol, move the CURSOR to the matching right symbol if it can be found on the in-core page(s). If the CURSOR is not under a left symbol, then search for the right symbol that returnes the nesting level to its current value. Note that this command is a NO-OP only when the desired level cannot be reached. If the end of the text is reached before a transition to the current level is found, the CURSOR is left in its original position and the deficiency is reported. Deficiencies may be of two sorts, either a left symbol is never reached, or a right symbol may never have been encountered while at the next nesting level.

⊗3⊗) As above but search for the right symbol that reduces the level from the specified nesting level, in this case from level 3 to level 2 (considering the current level to be 0).

⊗3⊗XRPAREN<CR> As above but not limiting the search to the in-core page(s).

⊗⇄ Undo the last parenthesis matching command and return the CURSOR to its former position. This allows one to look back and forth between the two matching symbols.

⊗XPAREN {<CR> Define the parentheses to be the symbols { and }. Note that a single space must separate the character from PAREN and only the left symbol of the set need be typed to select a set from the following list of character pairs:

( ) → ← ⊂ ⊃ ‘ ’ { } ‹ › [ ] .

⊗XPAREN Az<CR> Define the parentheses to be "A" and "z" (any pair of characters may be used). A distinction is made between upper and lower case letters. Note that both letters must be typed and that no extra spaces can be used (a space is a valid parenthesis!). Obviously, the scheme will not work if the chosen characters are used for other purposes in the text.

⊗XPINFO<CR> Type out information about the last parenthesis search.

9.7 Lisp Indenting Commands

E has limited ability to parse LISP S-expressions and to indent lines accordingly (with tabs and spaces as needed). Text is never moved from one line to another. Super-brackets are recognized, making] in the text close back to the matching [or close all hanging ('s if no hanging [is present to match. Except for this difference, the parsing is by MacLisp character set interpretation using a table of 128 entries that can be modified by the ⊗XLISPSYNTAX command (see EALS for details). The principal commands of interest are:

α/	Lisp indent the current line by context above it.
αβ/	Lisp indent the next line by context above it or the first attached line in context of the page.
α#α/	Lisp indent # lines starting with the arrow line.
⊗#αβ/	Lisp indent # lines starting with the next line or with the first line in the attach buffer in context of the page.

9.8 Issuing System Commands from E

There are a number of system commands that can be issued while in E. Some of the more important commands of this type are:

⊗XALIAS <project>,<programmer><CR> Change alias just as by the equivalent system command.

⊗XPROTECTION<CR> Type out the protection key for the file being edited.

⊗XPROTECTION nnn<CR>	Change the protection for the file being edited to the octal value nnn.
⊗XWRITTEN<CR>	Type the date and time when the present file was previously written and the PPN and jobname of the person who did it.
⊗XMAIL <destination and switches><CR>	Execute a system MAIL command using the text from the current page or attach buffer.
⊗0⊗XMAIL <destination><space><one line of text><CR>	Execute a system MAIL command, limiting the message to the one line of text that is typed between a space following the destination and the terminating <CR>, that is with no text from the page or attach buffer.
⊗XSEND <destination and switches><CR>	Execute a system SEND command using the text from the current page or the attach buffer.
⊗0⊗XSEND <destination><space><one line of text><CR>	Execute a system SEND command, limiting the message to the one line of text that is typed between a space following the destination and the terminating <CR>.
⊗XREMIND <destination and switches><CR>	Execute a system REMIND command.
⊗XDFIND <text><CR>	Start a FIND program as another job and report results. For detailed information on the FIND program see FIND.DON[UP,DOC].
⊗XSPOOL<CR>	Spool the contents of the current page or of the attach buffer.
⊗XXSPOOL<CR>	XGP spool the contents of the current page or of the attach buffer.

10. Macro Commands

Macros are provided in E to save the user the effort of retyping strings of commands that are needed frequently. A Macro represents any number of characters that will be given to E as commands and text (in lieu of this material being retyped) whenever the Macro is called by name or by a simple repeat command. A Macro may call other Macros or even itself (with nesting to a certain maximum depth), it may contain conditional branching and it may be called with repeat arguments. In fact, it is possible to write Macros that are, in effect, complete computer programs that will make major modifications to the file being edited.

All standard E commands (including LINE EDITOR commands) may be used within Macros. There is one very important restriction: if a αD or αI in a Macro is intended to be given at the end of a line, *it must be preceded by a α<TAB> or a αK<CR>.*

Macros may be defined and used during a single editing session and they may be written into a command file and saved for future use. Many users find it convenient to develop a series of Macros that serve their particular needs and to save them in a file in their own disk area under the name of EINIT.CMD. The contents of this file can then be loaded into the user's core area when needed during an editing session and they will then be available for repeated use. The file FINIT.CMD[1,3] contains several commonly used public Macros. An easy way to become familiar with the many intricate details of Macros is to copy one of these Macros and gradually modify it to meet one's particular needs.

Most users of E make but little use of Macros. While the subject is much too complex for adequate coverage in the present manual, an attempt will be made to show the value of the Macros and perhaps help new users to start using these commands. As a start, the user is urged to define a simple Macro, as explained in Section 9.1 below, and then to execute the Macro on a test program, using the commands listed in Section 9.2. Having done this, the next step might be to execute a Macro-defining command from the file FINIT.CMD[1,3], as explained in Section 9.3.

10.1 Defining Macros Directly

Macros may be defined with the command

⊗XDEFINE <Macro name><CR>.

The name may contain letters or digits (with only the first six characters being significant), it may be blank, or it can consist of exactly one of the following 9 characters:

≡ ∩ ∪ ⊂ ⊃ $ % ~ _

which appear on top of the digit keys on the DataDisc terminals. On receipt of this command, E asks the user to type the commands (and text strings, if any) that the Macro is to represent, ending with αβ<LINE>. The commands must be typed *in exactly the same way that they would be typed were they being given for immediate execution.* When the αβ<LINE> termination is typed, E will display the Macro name and its definition in terms of a standard notation (as given in the table in Section 9.6 below). The user of a DataMedia terminal may find the list in the Appendix useful to show him how these symbols will appear on the DataMedia screen.

Macros that are defined in this way are kept in the user's core area and they may be called for execution, as required, at any time during the same editing session. As explained below, Macros may be stored in a file for future use or they may loaded into the ATTACH buffer and then placed into the file that is currently being edited where they may be edited in the normal way before being again made available for execution or storage. Useful defining commands are:

⊗XDEFINE <Macro name><CR> Accept the following string of commands and text as a Macro under the name specified in accordance with the above explanation.

⊗0⊗XDEFINE<CR> Type out the names of all defined Macros.

⊗0⊗Z<Macro name><CR> Type out the named Macro without executing it.

⊗0⊗Y Type out the Macro that would be executed by the ⊗Y command as defined in section 9.2 below.

⊗XSET <newname>←<oldname><CR> Copy the definition of Macro oldname into the definition of a new Macro newname.

⊗XSET <newname>←<CR> Copy the definition of a Macro that was typed with a blank name into the definition of a new Macro newname.

10.2 Calling Macros for Execution

Macros are called for execution by one of the following commands.

⊗Z<Macro name><CR> Execute the named Macro once.

⊗3⊗Z<macro name><CR> Execute the named Macro 3 times.

⊗∩ Execute the special single-character Macro named "∩".

⊗8⊗∩ Execute the special single-character Macro named "∩" 8 times.

⊗Y Execute the last executed Macro or the last defined Macro if one has just been defined (but not considering any special single-character-named Macros).

⊗6⊗Y Repeat the execution 6 times of the last executed named Macro or the last defined Macro. Note that the symbol ∞ may be used for the number of executions if the Macro contains internal provisions for its termination.

10.3 Executing Commands and Defining Macros from a File

Commands stored in a file may be called for execution by the ⊗XEXECUTE command, as described below. It has become common practice to write Macro files as a series of ⊗XDEFINE<macro name><CR> commands, each followed by the command list that is being defined and a ⊗XSAY <macro name><CR> command (this command simply causes the macro name to be typed out). When such a file is called by the ⊗XEXECUTE command, it is the ⊗XDEFINE and ⊗XSAY commands that are executed. The defined Macro or Macros are simply defined (that is, loaded) into the user's core area without being executed and the user is given the names of the Macros so loaded. The user may then call the individual Macros for subsequent execution, as needed, using the commands listed in section 9.2.

⊗XEXECUTE <file name> (<page range>)<CR> Read the selected pages from the named file into a temporary Macro and then execute this Macro. If no file name is given, the default file will be used (initially EINIT.CMD on the user's log-in area). If no page range is given, the entire file is used.

⊗0⊗XEXECUTE<CR> Type out the current default execute file name.

10.4 Writing Macros into a CMD File

The command

⊗XPUTDEFS <file name><CR>

writes all of the Macro definitions that currently exist in the user's core area into the named file or into a default file, if no name is given. This default file will be the last file referenced by a ⊗XPUTDEFS<CR> or a ⊗XEXECUTE<CR> command, or if none has been referenced it will be the file EINIT.CMD on the users log-in file area. If the file chosen already exists, the newly stored Macros will be added to the file as a new page. If the file does not exist, one will be created. The Macros will be in the standard representation as listed in the left column of the table in section 9.6 below. Please note that these representations are *not* to be used when defining Macros with the XDEFINE command, although either of the two representations (or a mixture of the two) may be used if one writes or edits a Macro file for subsequent execution as described below.

10.5 Writing Macro Command Files Directly

Most users prefer to write complicated Macros as one would write any normal computer program, of course, using E as the editor. The alternate notation, as listed in section 9.6, simplifies this task, particularly for users of DataMedia terminals. A Macro file that is created in this way may contain a directory page (which will be ignored) and it may contain comments.

In command files, the character α is used to indicate that the CONTROL bit is to be added to the next command, and the character β that the META bit is to be added, quite analogous to the way these characters are used in this manual. However, the character ⊗ cannot be used to mean either α or αβ. Instead, it is used as an escape character to signify that the next character has a special meaning as shown in the following table. Note that the symbol ⊗ in a Macro must always be followed by one of the characters listed in the table. This is a situation where the α, β and ⊗ symbols are actually typed into the text when one writes the commands, in contrast with the situation where one is typing a Macro in directly, where the indicated <CONTROL> and <META> keys are used.

Extra spaces cannot be inserted into the command-file text for formatting as they will be taken to be part of the defined Macro. Tabs in the command-file text (as distinct from ⊗= or ⊗T) are ignored, however, and they may be used freely to separate commands or to set off comments from commands. Carriage returns and line feeds (not the commands ⊗⇄ etc.) also are ignored, allowing the text to appear on multiple lines for legibility.

10.6 Notation used in CMD files and in Typing Them

Representations		Command as accepted by E	
Standard	Alternate		
⊗=	⊗T	<TAB>	horizontal tab
⊗≠	⊗A	<ALT>	altmode
⊗⇄	⊗C	<CR>	carriage return
⊗←	⊗V	<VT>	vertical tab
⊗→	⊗F	<FORM>	form feed
⊗↓	⊗L	<LINE>	line feed
⊗↑	⊗B	<BS>	backspace
⊗α	⊗X	an α is wanted in the text	
⊗β	⊗Y	a β is wanted in the text	
⊗⊗	⊗⊗	a ⊗ is wanted in the text	
⊗;	⊗;	Ignore text from here to the end of the line (for comments)	
⊗⊂	⊗⊂	Ignore text from here to the next ⊃ (for long comments)	
α	⊗1	Add <CONTROL> bit to next command	
β	⊗2	Add <META> bit to the next command	

example: αF means the command <CONTROL>F
while α⊗F means the command <CONTROL><FORM>,
and α⊗→ means the command <CONTROL><FORM>.

10.7 Editing and Redefining Macros

It is often necessary to modify Macros, either to correct them or to adapt them for use in a slightly different situation. This can be done by using the following commands.

⊗XATTACH <Macro name><CR> Put the named Macro into the ATTACH buffer (which must be empty at the time this command is given). The attachment may then be deposited in the current file for editing in the normal way.

⊗XREDEFINE <Macro name><CR> Use the contents of the ATTACH buffer to redefine the named Macro.

10.8 Terminating and Resuming Macro Expansion

There are several commands available to the user to interrupt a Macro expansion. Only one of these is important enough to be reported here. For the others see E.ALS.

<ESC> Terminate the Macro expansion (either because it is doing the wrong thing or because you want to inspect what it is doing).

⊗XRESUME<CR> Resume the Macro expansion at the point where it was terminated.

10.9 Special Macro Commands

A full explanation of the many special commands available for use in Macros is beyond the scope of the present manual. It may be of interest, however, to list some of these so that the reader will have some idea as to what is available and will be better able to find wanted information in E.ALS.

The following commands are used to stop macro expansion in the event of an error and allow for the selective aborting of Macros that may have been called by other Macros

⊗XSTOPALL<CR> ⊗XSTOPONE<CR> ⊗XSTOPZERO<CR>

with ⊗XSTOPHOW<CR> being available to request a report as to which of the three is in effect.

The number-sign command in a Macro, ⊗# (where # is actually used in this case and does not mean some specific number, as it is used elsewhere in this manual) allows commands inside of a Macro to access and use the repeat argument given with the macro call.

The following commands may be used in macros and accept either a read-only-variable name or a macro name:

⊗XIFLT ⊗XIFLE ⊗XIFGE ⊗XIFGT ⊗XIFEQ ⊗XIFNE
⊗XADD ⊗XSUBTRACT ⊗XMAXIMUM ⊗XMINIMUM ⊗XMULTIPLY ⊗XDIVIDE
⊗XREMAINDER ⊗XARGUMENT ⊗0⊗Z.

The read-only variables available all end with a period and their names can not be abbreviated. Some typical ones are:

LINE. LINES. PAGE. PAGES. COL. COLS. COLINT. CHAR.
CHARS. CORLIN. CORLNS. CORCHS. ASCII. SIXBIT. CMAR. LMAR.
RMAR. BNUM. NFIND. NFOUND. NSUBST.

The commands, ⊗XSTEP <macroname><CR>, ⊗&, and ⊗XAUTOSTEP<CR> simplify debugging and allow the user to step through the execution of a Macro while seeing the results of each command.

Appendix A

Missing and Equivalent DataMedia Keys

Most of the missing keys on the DataMedia can be typed by holding the CTRL key and typing the character shown in the following table. Three characters require the use of 2 separate key stokes. Caution: do not confuse the α, β and $\otimes$, as shown in this table, with these symbols as employed in this manual to mean the use of control keys.

To Get	Hold and Type	then Type		Appears Brightened on Screen
↓	<CTRL>	a		a
α	<CTRL>	b		b
β		<NUL>	<CALL>	c
∧	<CTRL>	d		d
¬	<CTRL>	e		e
ε	<CTRL>	f		f
π	<CTRL>	g		g
λ	<CTRL>	h		h
<VT>	<CTRL>	k		k
<FORM>	<CTRL>	l	(or ↑L key only)	l
∞	<CTRL>	n		n
∂	<CTRL>	o		o
⊂	<CTRL>	p		p
⊃	<CTRL>	q		q
∩	<CTRL>	r		r
∪	<CTRL>	s		s
∀	<CTRL>	t		t
∃	<CTRL>	u		u
⊗	<CTRL>	v		v
⇄	<CTRL>	w		w
	<CTRL>	x		x
↑	<CTRL>	y		y
≠	<CTRL>	z		z
≤	<CTRL>	\		\
≥	<CTRL>	]		]
≡		<NUL>	<CLEAR>	
∨		<NUL>	<HOLD>	
<ALT>		<ESC>		
<BREAK>		<NUL>	− (minus sign)	
<BS>		<DEL>		
<CLEAR>		<MR>		
<ESC>		<NUL>		

Appendix B

File Switches

When entering a file, the user may request a number of options as expressed by switches. These switches consist of a slash (/), then an optional decimal number, then a single letter. Any number of switches may follow the file name. Should a conflict arise between switches, the later switch overrides the earlier one. Only a few of the possible switches will be described. For others see E.ALS[UP,DOC].

/C	Create a new file with the specified name. This is equivalent to the use of CET, as noted above. It can be used if one discovers that the C in CET was forgotten and one has just finished typing the file name. It may also be needed when switching files using the commands listed in section 3.1.
/F	Repage and reformat the file, limiting the maximum number of lines per page to the default value of 33. Existing page marks are retained so some pages may be shorter than the specified value. Reformatting causes the file to be rewritten on the disk with a possible saving in space by removing unnecessary null records at the end of each page.
/66F	Repage and reformat the file, limiting the maximum page size to 66 lines.
/∞F	The infinity sign may be used as a number. This switch does not introduce any extra page marks but it does remove any unnecessary null records, as noted above.
/Q	Quietly assume a yes answer to any question E might otherwise ask as to the need for reformatting, replacing an existing file, or discarding an invalid directory. Use this switch with caution: for example, the command CET FOO/Q<CR> will delete an entire file FOO, if it exists, and replace it with an empty file! Without the /Q switch, the user would be asked to verify the command before it would be executed.
/N	Edit the file without writing a directory on page 1. When this switch is used, E will create a new directory (on page 0) each time the file is called into core. The preferred way to use E is to allow a directory to be created on page 1 and to copy the file later omitting page 1, if necessary.
/F/R	Repage and reformat the core version only, without changing the disk copy. This is useful for examining an unformatted file with huge pages that has been produced by a program other than E.
/D	Search the system documentation disk areas for the partially named file and open in read-only mode the first possible file that is found. This same effect is produced by the monitor command READ, when used in place of ET. This command is of use when one is in doubt as to the exact name of some desired documentation file.
/33P/47L	Open the specified file on page 33 with the ARROW at line 47. If no page is specified, the first page (the directory) is assumed (or the text page, if there is only one), and if no line is specified, the first line is assumed.

Copy from fiche from IU (Bloomington)

This research was supported by the SAIL Computer Facility. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U.S. Government.

The picture on the cover of the aircraft carrier *U.S.S. Enterprise* is reproduced from an official U.S. Navy photograph as found in the collection from Plane Facts Company, New York, NY.